



Technische  
Universität  
Braunschweig



# Algorithmen und Datenstrukturen 2 – Übung #6

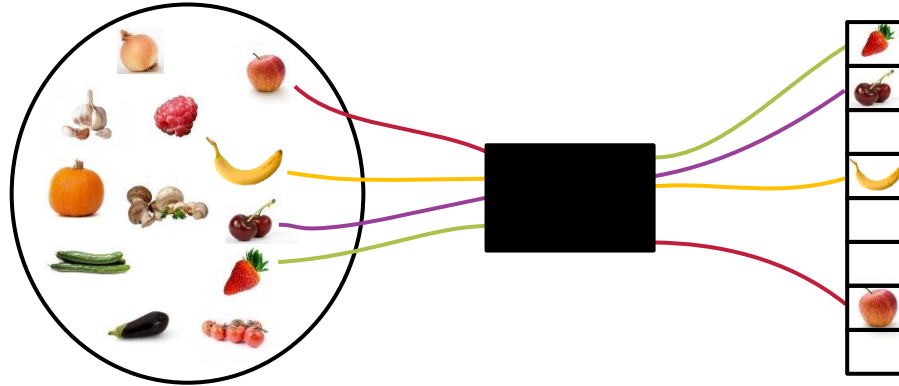
Hashing, Wiederholung

Matthias Konitzny

13.07.2022

# Heute

- Hashing
- Modulo
- Fragen/Wiederholung
- Dynamische Programmierung
- Approximation
- Reduktionen



# Hashing

# Motivation

Assoziative Datenstruktur mit extrem schnellen Operationen für Suchen, Einfügen und Löschen.

## Beispiel

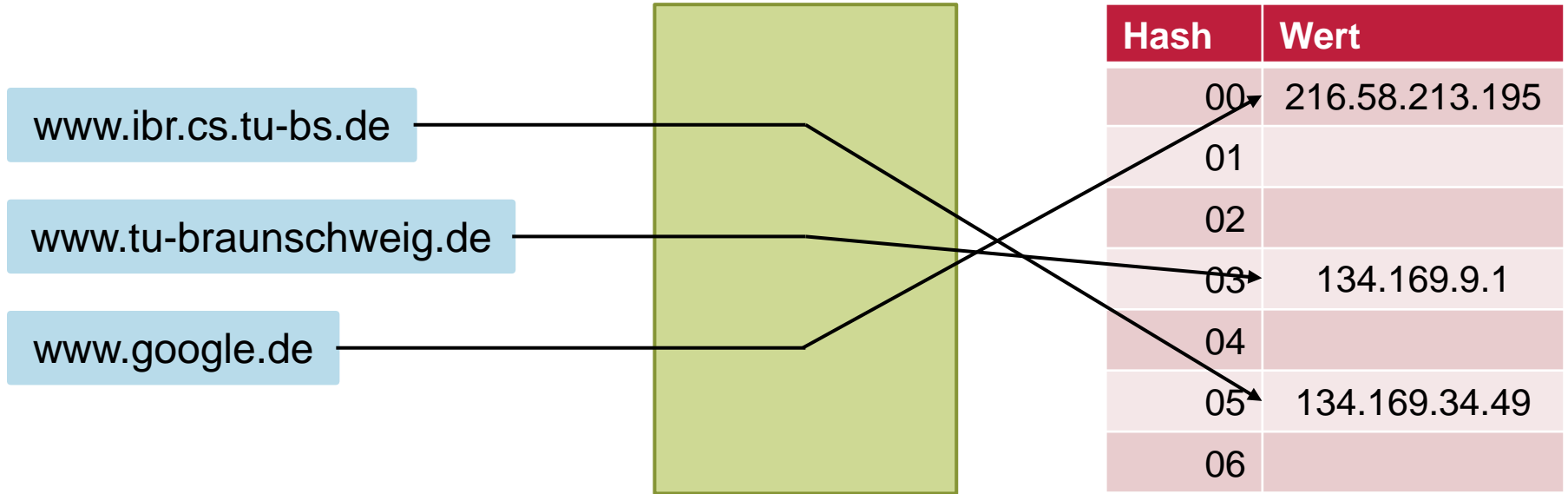
Domainnamen		IP-Adressen
www.ibr.cs.tu-bs.de	—————	216.58.213.195
www.tu-braunschweig.de	—————	134.169.9.1
www.google.de	—————	134.169.34.49

# Motivation

Schlüssel

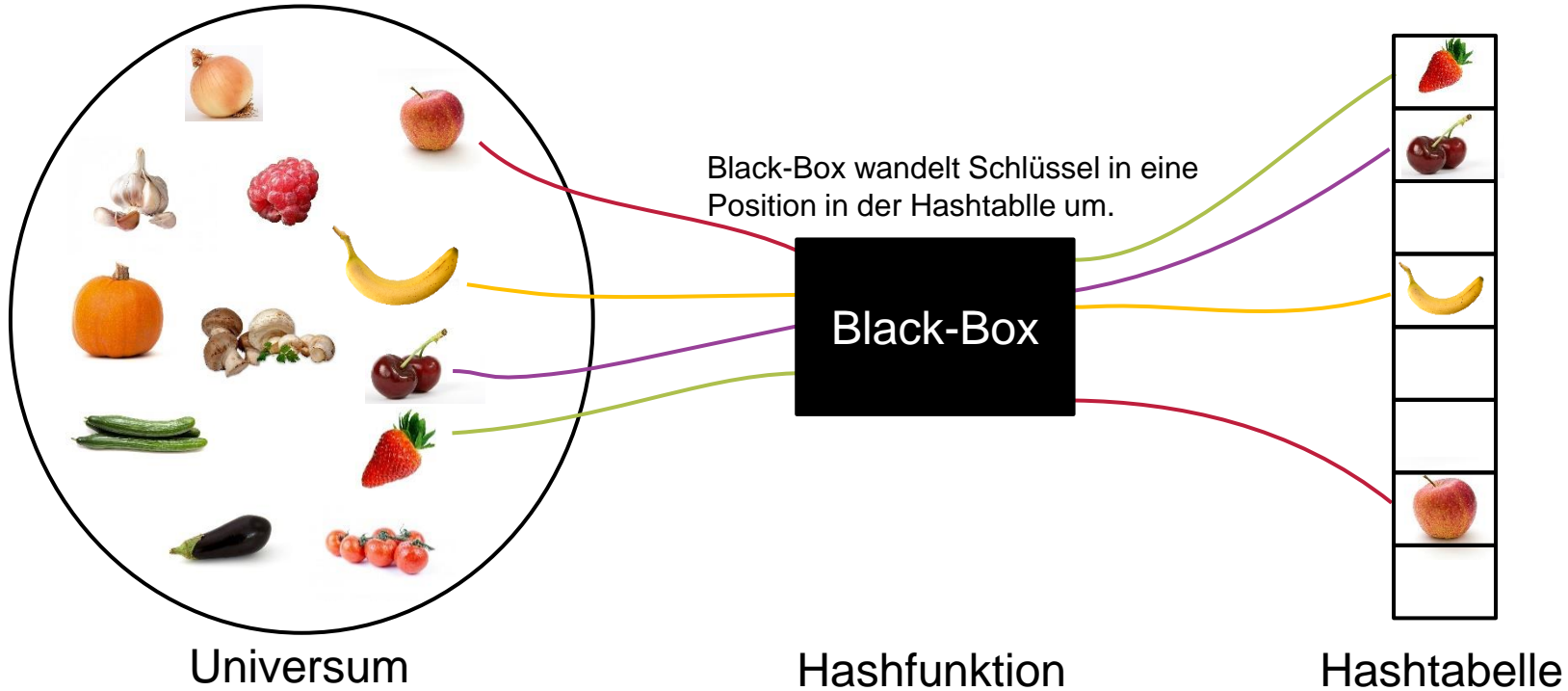
Hashfunktion

Hashtabelle



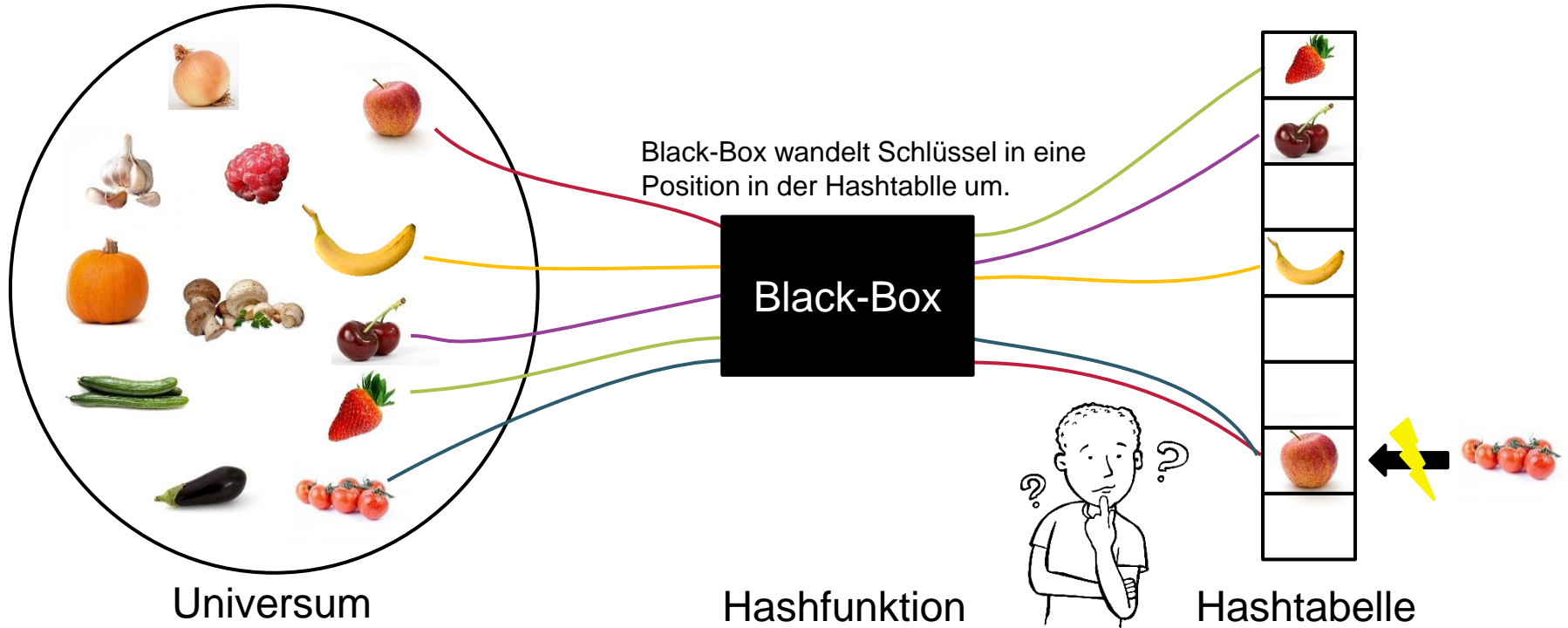
# Hashing

Jedes Objekt besitzt einen Schlüssel



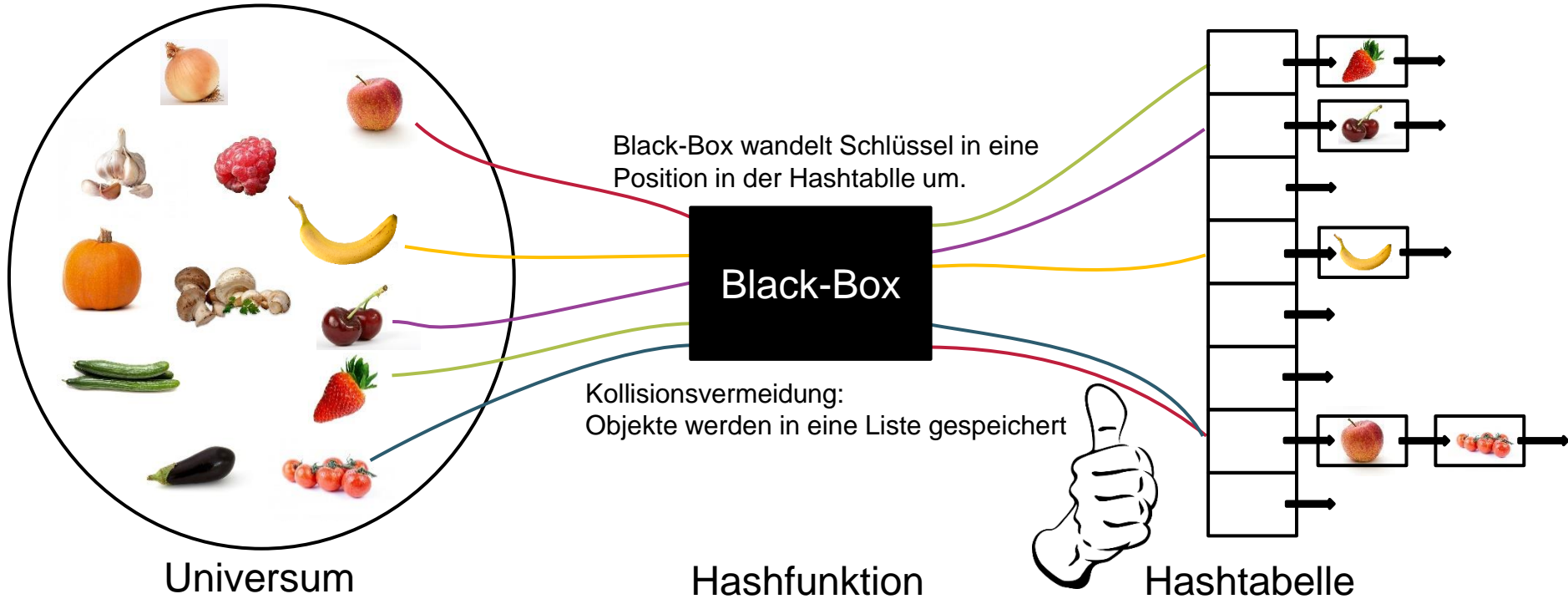
# Hashing - Kollision

Jedes Objekt besitzt einen Schlüssel



# Hashing - Listen

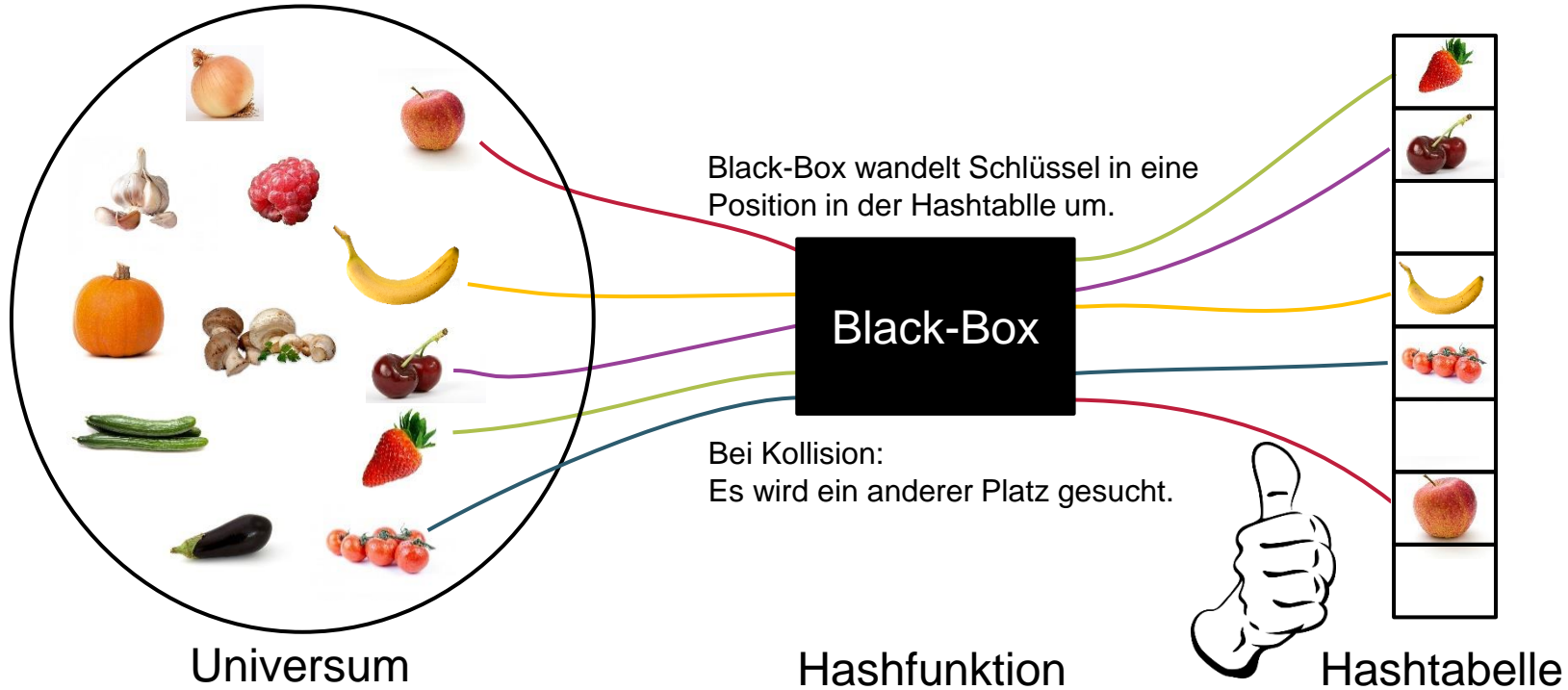
Jedes Objekt besitzt einen Schlüssel





# Hashing – Offenes Sondieren

Jedes Objekt besitzt einen Schlüssel



# Hashing Ohne Kollisionen

Sei  $P[h(x) = j] = \frac{1}{m}$ .

Wahrscheinlichkeit, dass  $k$  Elemente ohne Kollisionen eingefügt werden können:

$$P[k \text{ Elemente kollisionsfrei einfügen}] = \prod_{i=1}^k \frac{\# \text{freie Felder}}{\text{Größe Tabelle}} = \prod_{i=1}^k \frac{m - i + 1}{m}$$

Wahrscheinlichkeit, dass das nächste Element kollisionsfrei eingefügt wird, wenn  $n$  Elemente eingefügt wurden:

$$P[\text{nächstes kollisionsfrei}] = \frac{m - n}{m} = 1 - \frac{n}{m} = 1 - \beta$$

$\beta$  bezeichnet den *Belegungsfaktor*.

# Modulo – Restdivision

Für  $m \in \mathbb{N}$ ,  $x \in \mathbb{Z}$  und  $r \in \{0, \dots, m - 1\}$  ist

$$x \bmod m = r,$$

falls eine Zahl  $q \in \mathbb{Z}$  existiert, sodass gilt

$$x = q \cdot m + r$$

Damit ist auch für beliebiges  $i \in \mathbb{Z}$

$$(x + i \cdot m) \bmod m = x \bmod m$$

Beispiel:

$$27 \bmod 7 = 6$$

$$31 \bmod 9 = 4$$

Man kann zeigen:

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$

$$(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$$

# Hashfunktion – Offenes Hashing

Es sind

$$\begin{aligned} m &\in \mathbb{N}, \\ h_1: \mathbb{N} &\rightarrow \{0, \dots, m - 1\}, \\ h_2: \mathbb{N} &\rightarrow \{1, \dots, m - 1\} \end{aligned}$$

und

$$f: \mathbb{N} \rightarrow \{0, \dots, m - 1\}$$

Offenes Hashing benutzt eine Hashfunktion der Form

$$t(i, x) := (h_1(x) + f(i) \cdot h_2(x)) \bmod m$$

Beispiel für  $f(i) := (a \cdot i + b) \bmod m$  in rekursiver Schreibweise:

$$t(i, x) := \begin{cases} (h_1(x) + bh_2(x)) \bmod m & , \text{ falls } i = 0 \\ (t(i - 1, x) + a \cdot h_2(x)) \bmod m & , \text{ falls } i > 0 \end{cases}$$

# Hashfunktionen – Offenes Hashing

$$t(i, x) := (h_1(x) + f(i) \cdot h_2(x)) \bmod m$$

$t(i, x)$  nutzt

- **Lineares Sondieren**, falls  $f(i) \in \Theta(i)$  und  $h_2(x) \in \Theta(1)$
- **Quadratisches Sondieren**, falls  $f(i) \in \Theta(i^2)$  und  $h_2(x) \in \Theta(1)$
- **Doppeltes Hashing**, andernfalls

# Beispiel

$$t(i, x) = (4x^2 - 4 + (4x + 1)i) \bmod 13$$

Füge nacheinander die Zahlen 5, 7, 2, 6, 11 ein.

$$m = 13$$

$$t(0, 5) = 100 - 4 + 0 \bmod 13 = 96 \bmod 13 = 5$$

$$t(0, 7) = 4 \cdot 49 - 4 + 0 \bmod 13 = 4 \cdot 10 - 4 \bmod 13 = 10$$

$$t(0, 2) = 16 - 4 + 0 \bmod 13 = 12$$

$$t(0, 6) = 4 \cdot 36 - 4 + 0 \bmod 13 = 10$$

$$t(1, 6) = 4 \cdot 36 - 4 + 24 + 1 \bmod 13 = 1 - 4 + 11 + 1 \bmod 13 = 9$$

$$t(0, 11) = 4 \cdot 4 - 4 \bmod 13 = 12$$

$$t(1, 11) = 4 \cdot 4 - 4 + 44 + 1 \bmod 13 = 12 + 6 \bmod 13 = 5$$

$$t(2, 11) = 4 \cdot 4 - 4 + (44 + 1) \cdot 2 \bmod 13 = 12 + 12 \bmod 13 = 11$$

<i>i</i>	<i>T</i>
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

# Hashing in der Praxis

Auswahl der verwendeten Hashfunktion hängt stark vom Einsatzgebiet ab.

## Programmiersprachen



HashMap



dict()

## Sicherheit

*Prüfsummen*

*MD5*

*SHA*

*PGP*

## Spezialisierte Systeme

*Distributed Hash Tables*



*Chord*

# Semesterrückblick

## Zusammenfassung Algorithmen und Datenstrukturen 2



Matthias Konitzny | Übung #6 | Hashing, Wiederholung | Seite 17

## Wiederholung I Dynamische Programmierung



Matthias Konitzny | Übung #6 | Hashing, Wiederholung | Seite 24

## Wiederholung II Approximation



Matthias Konitzny | Übung #6 | Hashing, Wiederholung | Seite 29

## Wiederholung III Reduktion



Matthias Konitzny | Übung #6 | Hashing, Wiederholung | Seite 31

## Klausur



Matthias Konitzny | Übung #6 | Hashing, Wiederholung | Seite 37



# Zusammenfassung

## Algorithmen und Datenstrukturen 2

# Knapsackvarianten

Problem	0-1-KNAPSACK		MAXIMUM KNAPSACK				FRACTIONAL KNAPSACK
Typ	Entscheidungsproblem		Optimierungsproblem				Optimierungsproblem
Komplexität	NP-vollständig		NP-schwer				P
Algorithmen	DP	B&B	Greedy <sub>0</sub>	Greedy <sub>k</sub> ( $k > 0$ )	DP	B&B	Greedy
Bemerkung	-	-	Wert ist bel. schlecht	$1 - \frac{1}{k+1}$ -Approximation	Optimal	Optimal	Optimal
Laufzeit	$O(nZ)$	$O(n2^n)$	$O(n \log n)$	$O(n^{k+1})$	$O(nZ)$	$O(n2^n)$	$O(n \log n)$

Weitere Probleme:

- SUBSET SUM: NP-vollständig, Dynamisches Programm
- PARTITION: NP-vollständig, Dynamisches Programm
- BIN PACKING: NP-vollständig, lässt sich nicht besser als  $\frac{3}{2}$  approximieren.

# Hörsaal-Belegung

Problem	HÖRSAAL-BELEGUNG		
Typ	Optimierungsproblem		
Komplexität	P		
Algorithmen	Greedy Endpunkt	Greedy Kleinstes	Greedy Überlappung
Bemerkung	Optimal	$\frac{1}{2}$ - Approximation	$\frac{1}{2}$ - Approximation
Laufzeit	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Weitere Probleme:

- HÖRSAAL-AUSLASTUNG: P, Dynamisches Programm
- Färbung von Intervallgraphen

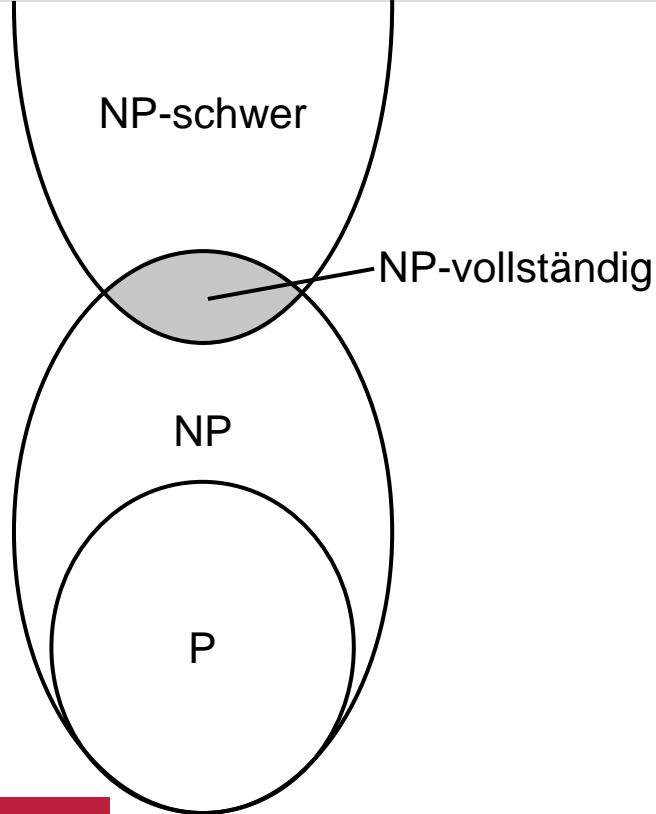
# Graphenprobleme

Problem	TRAVELING SALESMAN			MINIMUM VERTEX COVER
Typ	Optimierungsproblem			Optimierungsproblem
Komplexität	NP-schwer			NP-schwer
Algorithmen	Brute Force	B&B	DP	MaxMatching
Bemerkung	Optimal	Optimal	Optimal	2-Approximation
Laufzeit	$O(n!)$	$O(f(n)2^{n^2})$	$O(n^2 2^n)$	$O(n^2)$

Weitere Probleme:

- EUKLIDISCHES TRAVELING SALESMAN: NP-schwer

# Komplexitätsklassen



Ein Problem  $\Pi$  liegt in

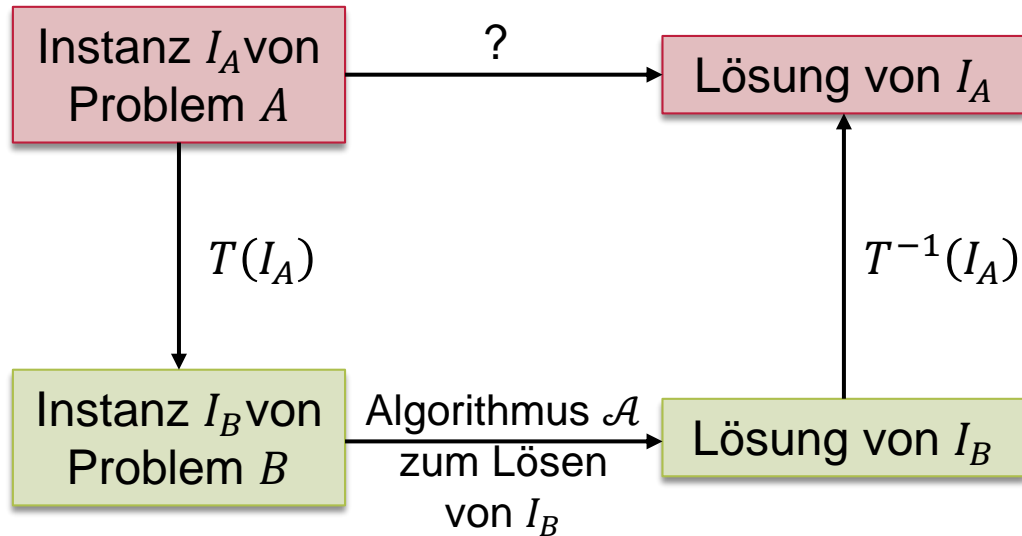
- **P**, falls es einen Polynomialzeitalgorithmus gibt, der  $\Pi$  korrekt löst.
- **NP**, falls jede Lösung in polynomieller Zeit verifiziert werden kann.

Ein Problem  $\Pi$  heißt

- **NP-schwer**, falls jedes Problem aus **NP** auf  $\Pi$  *reduziert* werden kann.
- **NP-vollständig**, falls  $\Pi$  **NP-schwer** ist und in **NP** liegt.

# Reduktionen

$T(I_A)$  und  $T^{-1}(I_A)$  besitzen polynomielle Laufzeit.  
Daher: Besitzt  $\mathcal{A}$  polynomielle Laufzeit, dann können wir die Lösung von  $I_A$  in polynomieller Zeit bestimmen.



# Was tun bei NP-Vollständigkeit?

	Heuristik	Approximation	Exakt
Vorteil	Sehr schnelle und oft sehr gute Lösungen.	Gibt Garantien für den Wert der Lösung.	Geben den optimalen Wert aus.
Nachteil	Kann beliebig schlecht sein	Oft nicht optimal, da der Worst-Case abgefangen werden muss.	Sie sind sehr langsam.
Beispiel	Greedy <sub>0</sub>	Greedy <sub>k</sub>	B&B, DP

# Wiederholung I

## Dynamische Programmierung





# Würfelsumme

Gegeben:  $n$  Würfel mit je  $m$  Seiten  
(und Werten  $1, \dots, m$ ) und eine Zahl  $K$ .

Gesucht: Anzahl Möglichkeiten den  
Wert  $K$  mit den Würfeln zu erzeugen.

Ein 6-seitiger Würfel:

$K$	0	1	2	3	4	5	6	7
#	0	1	1	1	1	1	1	0



# Würfelsumme

Zwei 6-seitige Würfel

<i>K</i>	0	1	2	3	4	5	6	7	8	9	10	11	12	13
#	0	0	1	2	3	4	5	6	5	4	3	2	1	0

Summe!



Drei 6-seitige Würfel

<i>K</i>	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
#	0	0	0	1	3	6	10	15	21	25	27	27	25	21	15	10	6	3	1	0



# Würfelsumme

**Function**  $A(K, n)$

$A[0..nm][0..n]$  //2D Array

$A[0][0] := 1$

**for**  $i = 1$  **to**  $K$  **do**

$A[i][0] := 0$

**for**  $j = 1$  **to**  $n$  **do**

**for**  $i = 0$  **to**  $j - 1$  **do**

$A[i][j] := 0$

**for**  $i = j$  **to**  $jm$  **do**

$k := \min(i, m)$

$A[i][j] := \sum_{\ell=1}^k A[i - \ell][j - 1]$

**for**  $i = jm + 1$  **to**  $nm$  **do**

$A[i][j] := 0$

**return**  $A[K][n]$

$$A(i, j) := \begin{cases} 0 & , \text{falls } j = 0, i > 0 \\ 1 & , \text{falls } j = i = 0 \\ 0 & , \text{falls } i < j \\ 0 & , \text{falls } i > mj \\ \sum_{\ell=1}^{\min(i, m)} A(i - \ell, j - 1) & , \text{sonst} \end{cases}$$

# Wiederholung II

## Approximation

# Approximation – Probleme mit Parameter

Manchmal besitzt man Informationen über die Instanzen.

Beispiele:

**MAXIMUM KNAPSACK** mit  $\alpha \in \mathbb{N}$

Für alle  $i \in \{1, \dots, n\}$  gilt  $z_i \leq \frac{Z}{\alpha}$

→ GREEDY<sub>0</sub> ist eine  $\frac{\alpha-1}{\alpha}$  - Approx.

**BIN PACKING** mit  $\alpha \in \mathbb{N}$

Für alle  $i \in \{1, \dots, n\}$  gilt  $z_i \leq \frac{Z}{\alpha}$

→ FIRST FIT ist eine  $\frac{\alpha}{\alpha-1}$  - Approx.

**SET COVER** mit  $k \in \mathbb{N}$

Jedes Element  $u \in U$  kommt in maximal  $k$  Teilmengen in  $\mathcal{F}$  vor.

→ Es gibt eine  $k$  - Approx.

Für  $k = 2$  ist das  
VERTEX COVER.

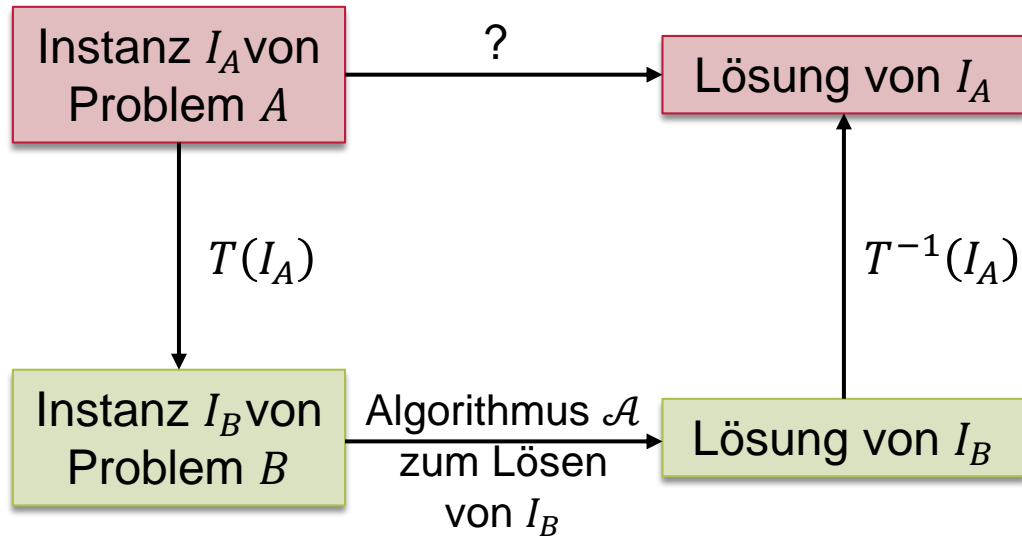
# Wiederholung III

## Reduktion

# Reduktionen

$T(I_A)$  und  $T^{-1}(I_A)$  besitzen polynomielle Laufzeit.

Daher: Besitzt  $\mathcal{A}$  polynomielle Laufzeit, dann können wir die Lösung von  $I_A$  in polynomieller Zeit bestimmen.





# 3SAT-3

**Gegeben:** Formel wie bei 3SAT, aber jede Variable kommt in maximal drei Klauseln vor.

**Frage:** Lässt sich die Formel erfüllen?

Dieses Problem ist NP-schwer!

Wir zeigen:  $3SAT \leq_p 3SAT-3$

Problematisch nur Variablen, die öfter als drei Mal vorkommen.

Wie können wir das auflösen?

# 3SAT-3

$$(\overset{\circ}{x_i} \vee x_j \vee x_k) \quad (\overset{\circ}{\bar{x}_i} \vee \bar{x}_o \vee x_q) \quad (\overset{\circ}{x_i} \vee \bar{x}_p \vee x_j) \quad (\overset{\circ}{x_i} \vee x_p \vee \bar{x}_o)$$

Für jedes der  $n_i$  Literale von Variable  $x_i$ :

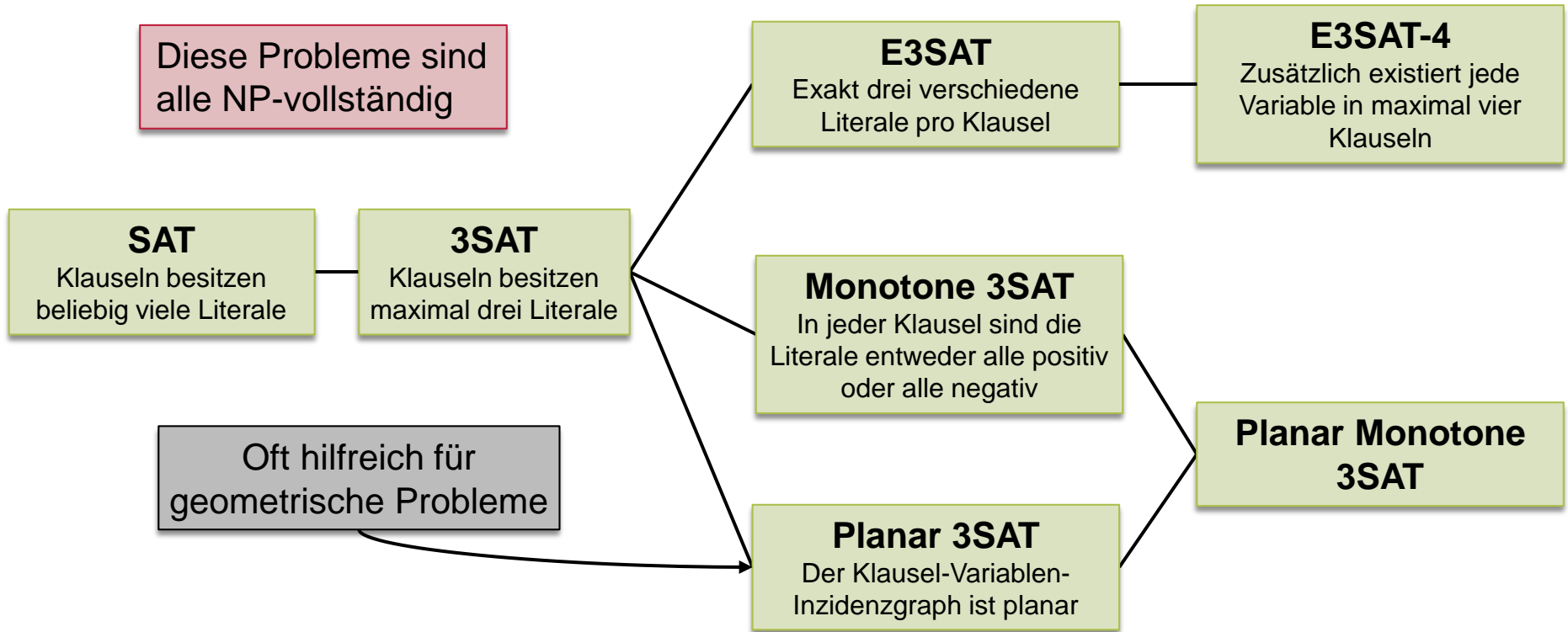
- Erzeuge Variablen  $x_{1,i}, \dots, x_{n_i,i}$ .
- Ersetze das  $c$ -te Literal von  $x_i$  mit einem Literal der Variablen  $x_{c,i}$ .
- Füge Klauseln der folgenden Form hinzu.

$$(x_{1,i} \vee \bar{x}_{2,i}) \wedge (x_{2,i} \vee \bar{x}_{3,i}) \wedge \dots \wedge (x_{n_i,i} \vee \bar{x}_{1,i})$$

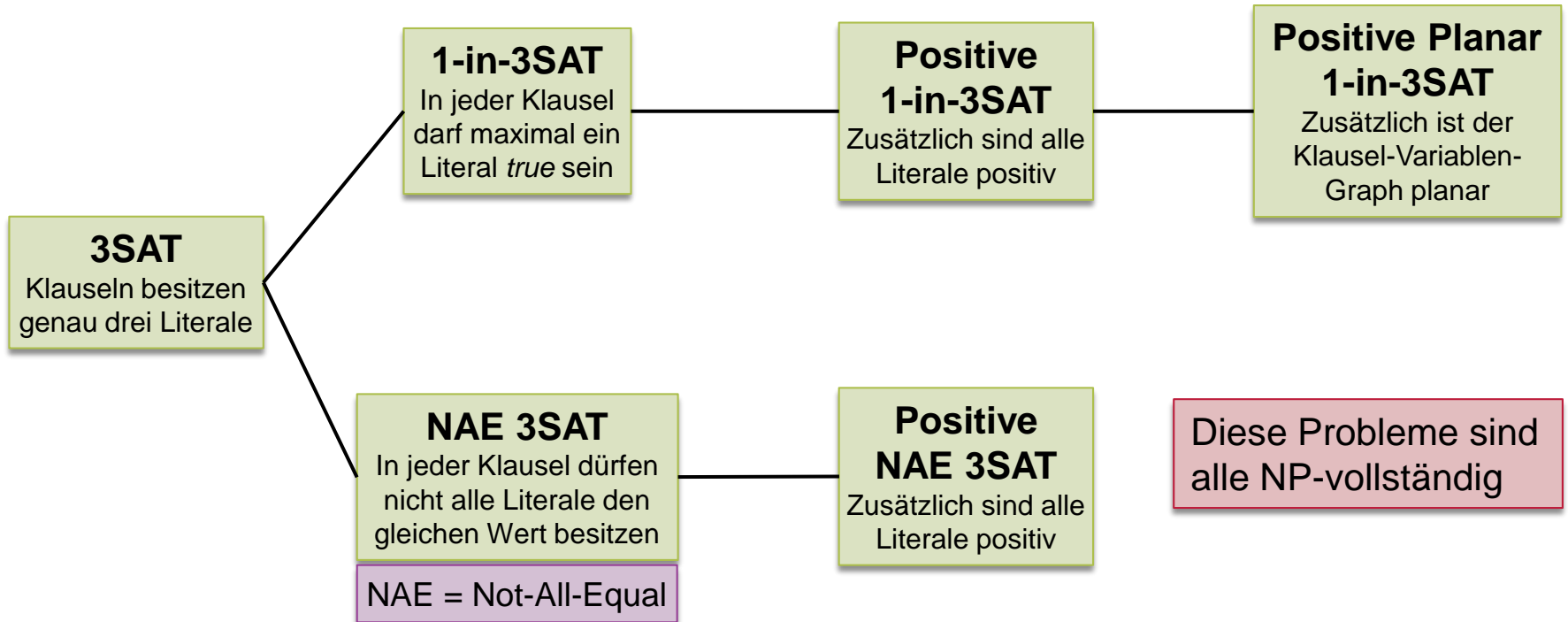
Beobachtungen:

- Die neuen Variablen tauchen genau drei Mal auf.
- Ist eine Variable auf *true* gesetzt, sind alle *true*. (Repräsentieren die gleiche Variable!)
- Die alte Variable taucht nicht mehr auf.

# Weitere 3SAT-Varianten



# Noch mehr 3SAT Varianten



# Klausur

# Klausurinfos

## Prüfung Informationen

Allgemein / Von konitzny

Liebe Studierende,

die Prüfung zu AuD2 findet am 19.08.22 statt. Nachfolgend stehen Informationen zur Prüfung.

## Informationen

- Beginn: 12:30 Uhr
- Raum: UP 3.007 (A-N), SN 19.1 (O-Z) (getrennt nach Nachname; 1. Buchstabe)
- Dauer: 2 Stunden
- Bitte 15 Minuten früher anwesend sein.
- Mitbringen:
  - Studierendenausweis (plus Lichtbildausweis, falls kein Foto vorhanden),
  - Dokumentenechter Stift (kein Bleistift, kein rot)
  - Wörterbuch, falls nötig
- Weitere Unterlagen sind nicht erlaubt.
- Eigenes Papier ist nicht erlaubt; wir stellen Papier

Was man wissen sollte:

- Alles aus der Vorlesung (Ohne höherdimensionales Packen)
- Übersicht über die Übungen/Hausaufgaben